(12)

Clarification Dialogues: A Step Toward-Semantic-Level Interaction Paradigms

Peter Aberg and Robert T. Neches
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

October 1992
ISI/RR-93-341

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1992	Research Report

4. TITLE AND SUBTITLE
Clarification Dialogs: A Step Toward Semantic-Level Interaction Paradigms

5. FUNDING NUMBERS
NCC2-719 and
N00174-91-0015

6. AUTHOR(S)
Peter Aberg and Robert Neches

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
RR-341

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Interface designers typically think of "dialogs" in terms of low-level, relatively syntactic interactions such as menu selections. When trying to make systems with complex information spaces more accessible to users, however, providing graphical user interfaces that deal mainly with such low-level presentation concerns is only part of the solution. Of greater importance is the use of interaction paradigms based on a notion of well-defined, higher-level dialog games in human-to-human interaction, facilitate communication by making it clearer what capabilities each side is expected to have, the kinds of input they expect, and how they can be expected to have, the kinds of input they expect, and how they can be expected to interpret and respond to those inputs. In this paper we describe how a particular interaction paradigm of this kind, called specification by reformulation, can be seen as a clarification dialog, one form of dialog game. We focus on presenting the procedure involved in analyzing new application domains in order to instantiate this paradigm within them, and exemplify with an implementation of a tool to assist users in selecting reports from a very large database system.

14. SUBJECT TERMS
User dialogs, user interface paradigms, complex information spaces, clarification dialogs,dialog games, specification by reformulation, retrieval by reformulation

15. NUMBER OF PAGES
8

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

CLARIFICATION DIALOGS: A STEP TOWARD SEMANTIC-LEVEL INTERACTION PARADIGMS

Peter Aberg and *Robert Neches*

University of Southern California, Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
(310) 822-1511
aberg@isi.edu; neches@isi.edu

ABSTRACT

Interface designers typically think o[illegible]" in terms of low-level, relatively syntactic inte[illegible]ch as menu selections. But when trying to [illegible]stems with complex information spaces more [illegible]le to users, providing graphical user interfaces that deal mainly with such low-level presentation concerns is only part of the solution. Of greater importance is the use of interaction paradigms based on a notion of well-defined, higher-level dialogs. Such paradigms, analogous to what linguists have called dialog games in human-to-human interaction, facilitate communication by making it clearer what capabilities each side is expected to have, the kinds of inputs they expect, and how they can be expected to interpret and respond to those inputs. In this paper, we describe how a particular interaction paradigm, specification by reformulation, can be seen as a clarification dialog, one form of dialog game. We focus on presenting the procedure involved in analyzing new application domains in order to instantiate this paradigm for them, and exemplify it with an implementation of a tool to assist users in selecting reports from a very large database.

KEYWORDS: Clarification dialogs, user dialogs, user interface paradigms, complex information spaces, specification by reformulation, retrieval by reformulation, dialog games.

INTRODUCTION

A large body of recent work on human-computer interaction focuses on means of making complex information systems, such as very large databases or computer networks, accessible to relatively inexperienced users (see for instance [4, 2, 1, 3, 9]). The two major problems novice users of these systems are faced with are those of mentally grasping the scope and structure of the information spaces presented by the systems, and of formulating commands to navigate through such information spaces to locate items of interest [4]. These problems are not adequately addressed solely by building easy to use, graphical user interfaces to the systems; such interfaces are only part of the solution. Of greater importance than focusing on low-level details such as button layout conventions and menu organizations, is the use of interaction paradigms that emphasize the notion of high-level dialogs in human-computer interaction.

RABBIT [9], HELGON [3], and our own previous research on the BACKBORD system [11] have demonstrated the viability of the *retrieval by reformulation* paradigm to this end. In retrieval by reformulation, users find items they are searching for in a database by successively refining a partial description of the items, guided by feedback provided in the form of example items matching the description as it appears at each iteration. In this paper, we argue that retrieval by reformulation represents an instantiation of a general style of naturally occurring human dialog that characterizes interactions intended to help clarify intent [6], and that it should be possible to apply this paradigm to a wide range of application domains beyond database retrieval. Other authors, such as Stelzner and Williams [7] and Yen [11], have already demonstrated that retrieval by reformulation can be generalized to be used in applications such as knowledge based interface development and electronic mail message creation. Stelzner and Williams coined the term *specification by reformulation* for this, more general version of the paradigm. The common denominator among these applications, as well as the RABBIT and HELGON systems, is that their human-computer interaction component can be expressed in terms of a *clarification dialog*. Generally speaking, the role of the computer in such a dialog is to ask users, "Is this what you mean?" at each dialog iteration; to which users respond either affirmatively, or by providing further inputs to clarify their intentions, guided by feedback provided by the system. On-line help system navigation, interactive database query generation, and document library browsing are a few examples of situations where such a dialog can occur.

Our main focus in this paper is on describing the general process of analyzing an application to understand how to instantiate a clarification dialog within a new domain, expressed in terms of the specification by reformulation paradigm. We illustrate this process for a particular case by presenting our work on the design and implementation of BB386 (the name signifies it is a derivative version of BACKBORD that runs on any 386-based, or better, PC), a system to help users select reports from very large databases. We precede our analysis section with a discussion of the essential components of the specification by reformulation paradigm, and related work, in the light of clarification dialogs. We close by generalizing the lessons learned from implementing BB386 to emphasize the importance of designing user interfaces that support the notion of high-level dialog paradigms.

CLARIFICATION DIALOGS

The variations in skill levels between different users of complex information systems, such as very large databases, can be expressed in terms of the degree of *metaknowledge* [1] they possess about data stored in the systems. Expert users know where to look for something, how it is represented in the system, and how to formulate a request to retrieve the data. Inexperienced users, on the other hand, might have only vague ideas of what they want to find, and little or no knowledge of how to go about communicating their intentions to the system. This gap is primarily of semantic nature, and cannot be bridged simply by providing sophisticated user interfaces to such systems. What is needed instead is support for a notion of human-computer dialogs at the *conversational* level (although not necessarily expressed in terms of natural language interaction), instead of at the level of describing window appearances, etc. User interface paradigms based on high-level dialogs are capable of bridging the semantic metaknowledge gap by providing users with several forms of feedback and guidance as they work. The very successful retrieval by reformulation paradigm is an example of this. When coupled with well-designed, graphical user interfaces, such paradigms can also bridge any syntactical gap that may be present. Later in this paper we present the steps involved in analyzing the semantics of a general application domain to instantiate a high-level dialog based paradigm for its user interface. First, however, we turn to the origins of our notion of high-level dialogs, in regular, human dialog.

Dialog Games

In their research on naturally occurring human dialog, Mann [6], and Levin and Moore [5], found that people appear to interact according to mutually agreed upon patterns, or rules, that constrain their roles and behaviors as participants in a dialog. These rules are organized around the goals each participant seeks to attain through the dialog. Mann defined the term *dialog games* for such goal-oriented exchanges, and provided a partial classification of some of the commonly occurring dialog games. He found that the notion of dialog games plays a vital part in correct interpretation of dialogs and dialog components. Indeed, it is often the case that individual sentences of a dialog are impossible to interpret correctly without knowing what dialog game they are part of. For instance, the question, "What's the current CPU load factor?" can be interpreted either as a request for the actual load factor expressed as some numeric value, or as a request for an explanation of what the term "current CPU load factor" represents. Which interpretation is the correct one depends on the current dialog game. If the question is posed as part of an *information-seeking* dialog game (see below), the first response is the correct one. If it is posed as part of a *helping* dialog game (see below), the second one is correct. A dialog game only works if both participants act according to the conventions of that particular game; i.e., are 'playing the same game'.

In many computer application domains, such as database retrieval or command selection, the user interaction closely resembles one of the three dialog games (as defined by Mann): *helping*, where the initiator wants to solve a problem and interacts with the respondent to arrive at a solution; *action-seeking*, where the initiator wants some action performed and interacts with the respondent to get him to perform it; and, *information-seeking*, where the initiator wants to know some specific information and interacts with the respondent to learn it. Although computers are a long way from being able to participate in the free-form dialogs Mann et. al. studied, it is still useful to try to apply the underlying insights to human-computer interactions by creating analogous interaction paradigms. Building applications that make clear what dialog games they support allow users to establish a degree of confidence in the systems: They can expect such systems to behave in certain ways, and to support certain kinds of operations appropriate to the dialog games. The actual process involved in designing the user interface component of such applications is illustrated in the section "Problem Analysis," later in this paper.

Retrieval by Reformulation

It is interesting to view retrieval by reformulation systems, such as RABBIT [9] and HELGON [3], in the light of dialog games. These systems help users create database (or knowledge base) queries by successively refining a partial description of the items sought after. The retrieval by reformulation paradigm was derived from a psychological theory of human remembering that states that examples and other associations are more important to people than formal attributes when defining categories of items [10]. Both RABBIT and HELGON consist of three major components: a *partial description*, a *list of example items* matching the description, and a *view of one example item*. Users begin either with a very general description that matches all items in the database (the only alternative supported by RABBIT), or with a description of some item they already are familiar with (HELGON supports both

alternatives). This initial description is transformed by the system into a database query, which is then used to retrieve example items consistent with it. Users can view individual examples, and select portions of them to incorporate back into the description, thus reformulating it to better express their intent. The refined description is then used to retrieve new matching items. The entire sequence is repeated over again, until the desired items are finally retrieved. This process can be seen as a form of the information-seeking dialog game in which the system's role, in effect, is to ask users, "Is this what you mean?" questions at each step of the way. Users may respond to these questions by answering, in effect, "Yes!" or "No. I want to add this component too [pointing to component in example]," or "No. Generalize this value [pointing to value in description]," and so on. We refer to this cooperative process as a *clarification dialog*. The three components of the RABBIT and HELGON systems, the partial description, the matching examples list, and the presentation of individual examples, respectively embody the actions of *refinement*, *feedback*, and *guidance*, key parts of a clarification dialog.

Users working with RABBIT or HELGON are navigating through an information space where each node in the space is represented by a description. The examples retrieved at each step along the way provide guidance by showing users both what terms they can refer to in the description, and how the information space is structured. This explorative approach allows users to rapidly focus their search on smaller, more manageable subsets of the information space. The two systems take different approaches to determining in what way users can move through the information space, however. In RABBIT, user input is limited to critiquing the examples retrieved by the system. This ensures that users can only refer to objects they have seen, and thus are known to exist in the database. In HELGON, it is possible to refine the description either by using examples, or by directly typing in values. This allows experienced users, who know what they want, to say so directly, thereby skipping several steps in the reformulation process.

Specification by Reformulation

Our initial implementation of the BACKBORD system [11] was similar to HELGON in terms of how users could seed the reformulation process. BACKBORD allowed users to start with either a very general description and refine it by adding further details, or with a description of some known item that could then be generalized, or in other ways altered, to produce the desired results. The methods available to refine the description were more akin to RABBIT than HELGON, however. Our application domain for BACKBORD went beyond generic retrieval and editing of databases and knowledge bases, leading us to use to join Stelzner and Williams [7] in using the term *specification by reformulation* to describe our user interaction paradigm. In specification by reformulation, no assumptions are made about the kinds of objects the partial description refers to, hence the method of generating feedback for users could be through database retrieval, simple table lookup, numerical computation, intelligent reasoning processes, etc. The range of possible application domains is very large, and could include database query construction, operating system command selection, report selection, or on-line help system navigation.

Specification by reformulation is a more general instance of the information-seeking dialog game. The three important actions of a cooperative clarification dialog, those of refinement, feedback, and guidance, are retained, as are their respective embodiments in the partial description, the list of matching example items, and the example item view. The paradigm thus retains all the advantages of retrieval by reformulation when dealing with complex information spaces, while adding flexibility in terms of the structure and contents of the partial description. This entails a need for two additional components to the paradigm, however: one to process the partial description before it can be used to generate feedback, and another to determine what parts of the feedback information to display, as well as how to display it. The actual implementations of these components may be highly application specific, but they will share the same, core set of semantic relationships.

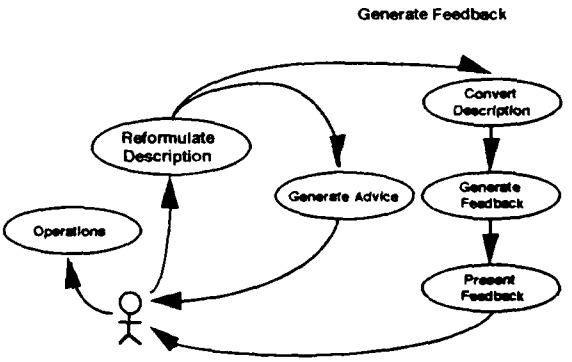

Figure 1: Refinement Loop of the Specification by Reformulation Paradigm

Figure 1, from [11], illustrates the refinement loop of the specification by reformulation paradigm and the relationships between its components. There are two separate, complementary feedback loops shown in the figure. The innermost loop mainly provides advice to the user on how to proceed with the description, such as what attributes are still missing from it for it to qualify as being complete (where completeness is defined by the application domain). This advice can be of both syntactic and semantic nature. For instance, for an electronic mail message to be complete, it must have a recipient, a subject, and a message body.

The outermost loop provides feedback in the form of results produced when the partial description created by the user is applied to the application's information space. The description is first converted into a form suitable for generating feedback. This conversion process may, for instance, entail translating a graphical representation of the description, as expressed by the user, into a declarative form that can be processed by the system. Or, in some applications, it may mean the relaxing of constraints specified by the user, in order to generate a broader spectrum of feedback items. The next step of the loop is to generate the actual feedback. As mentioned earlier, this can be through any one of several methods depending on the application domain. Once the feedback has been retrieved, the final step is to process and present it to the user. This processing can entail filtering out unnecessary items, or parts of items. How to actually present the feedback items is determined by the application domain, and might also be influenced by the types of the items themselves.

Instantiating the specification by reformulation paradigm for an application requires mapping the components shown in Figure 1 onto elements of the application dialog. The steps involved in this process are presented in the following section.

PROBLEM ANALYSIS

The Air Force Logistics Command (AFLC) operates several database systems as part of their efforts to maintain inventories of parts for the Air Force (where a part can be anything from a bolt, to a jet engine, to an entire airplane). Each of these databases deals with different aspects of the parts procurement, or inventory management, process. For example, one database holds information on purchase contracts and the line items of such contracts, another holds detailed information on various parts and their current availability, etc. These databases are used by people in positions ranging from buyers to upper-level management. For each database there are a number of reports available that users may request. Each report takes one or more parameters as input, and returns a printout in a particular format (on paper or on a terminal) of the information requested. The total number of available reports numbers in the upper hundreds. These reports are completely independent of each other and are not organized into any form of hierarchy or other structure. For instance, it is not possible to refer to, as a group, reports that share certain characteristics. Users are not able to generate their own, custom reports to the databases (for reasons of security, as well as various system limitations).

High Level Analysis

If we are considering instantiating the specification by reformulation paradigm for an application, we must first determine if our application matches the characteristics of domains in which this paradigm is useful. Specification by reformulation is designed to operate on an information space where each node in the space can be represented by some form of description, and where there exists a notion of relationships between nodes that is captured in these descriptions. The relationships need not be explicitly present in the information space in the form of links or hierarchical structures; as long as they can be captured in the descriptions. The tasks of users working with such an information space must be capable of being expressed in terms of navigating through the information space to locate nodes (items) of interest.

Looking again at the AFLC's systems, we find that they do present a large, complex information space to their users, where the nodes of the information space represent the available reports. The relationships between nodes (reports) is purely implicit (i.e., there are no explicit representations of them in the system, but such relationships do exist, for instance, in terms of what information different reports return). The task facing users of these systems is to navigate through the information space to find reports they need. Clearly, the characteristics required by the specification by reformulation paradigm are all present.

To instantiate specification by reformulation for this application, we need to determine the elements of the application dialog and how these can be mapped onto the components of the paradigm. There are six basic steps to performing this mapping, as presented below. Refer to Figure 1 for an illustration of the relationships between each step.

- *Establish the topic of the partial description.* What is it users are trying to describe? Some thought also has to be given to a possible formal representation of the description topic.
- *Determine what operators are relevant to refining the description.* The paradigm defines a few generic operators for modifying descriptions, such as adding or deleting parts, and if the application's information space is suitably structured, *specialize part* and *generalize part* . We need to see how, and if, these can be instantiated, and then determine what domain-specific operators are needed as well.
- *Decide what kinds of feedback can be generated based on the partial description.* As a minimum we want to have a list of items matching the description. Other kinds of possible feedback include different forms of advice (see Figure 1).
- *Determine how to convert the description to a form suitable for generating feedback.* This might mean constructing a database query from the description, for instance.
- *Decide how feedback should be presented.* Is there a need for filtering the feedback, or for converting it into another form? Numerical data may, for instance, be converted into graphs in this step.

- *Determine in what form user guidance can be provided from the feedback.* Allowing users to look at, and work with, example items returned as feedback is one form of guidance. There can also be other forms, such as syntactic guidance – so that only syntactically correct descriptions can be generated.

Instantiating the Paradigm

We will use AFLC's database systems to illustrate this process:

Establishing a description topic: Since the nodes of this application's information space are the individual reports, these should be the subject of the refinement process, and therefore the topic of the partial descriptions users will work with.

Determining relevant operators: We will postpone this issue until we have decided on a formal representation for the description (see the section "Formal Representations," below).

Determining kinds of available feedback and advice: Referring to Figure 1, if we begin with the outer loop, the feedback we want is a list of reports matching the partial description. But, in this case, the information space we are working with does not provide any support for retrieving reports based on their attributes (reports can only be retrieved by name). Since we cannot generate our own custom report requests to the databases (for reasons mentioned earlier), there are two alternative approaches to solving this problem. The first is to retrieve all reports from the databases and match them, one by one, against the description – an impractical solution that would be very expensive in terms of computing time. The second is to build a knowledge base, internal to our system, that models each report using a formal representation that can easily be matched against the description. This approach is much faster than the first alternative, but it does require that a knowledge base be constructed containing entries for every available report. It also requires maintaining this knowledge base as new reports are added, and old ones removed. Nevertheless, this seems like a small price to pay considering our other alternative. It also has the advantage of making the next step trivial, since similar representations of reports can be used for the description and the knowledge base.

In the inner feedback loop of Figure 1, we generate advice to the user. One such category of advice is informing users about valid completions of the current description. Having this advice would help constrain users' search through the information space. Since, in our application, the entire information space is modeled in the internal knowledge base, such advice is easily generated.

Determining need for description conversion: In this case, due to the knowledge base representation we chose in the previous step, this step is trivial.

Deciding how to present feedback: Showing a list of the names of reports that match the partial description is an adequate solution for this application.

Determining forms of guidance to provide: Adhering to the specification by reformulation paradigm, guidance would best be provided by showing examples of output generated by each report that matches the partial description. Unfortunately, a report requires certain parameters to execute, and it would be impractical to force users to provide such parameters to simply see a sample of a report's output. To solve this problem, we can use 'canned' report results. For each report, we 'can' one or more typical report outputs to give users an idea of what they look like. The refinement loop is completed by allowing users to select fields from the report and introduce them into the description.

Defining the Representation

With our high level analysis complete, we now turn to developing a representation for the reports. This representation must capture the distinct features of each report and allow us to talk about similarities, and other relationships, between them. It is also the terminology users will employ when constructing descriptions of reports they seek to find. Development of a formal representation is of such vital importance to users' acceptance of the completed system that user participation in this process is crucial. The representation we arrived at, in close cooperation with AFLC, decomposes a report into four components: a *subject*, basically the main topic of the report; one or more *attributes*, providing further detail to the subject; a *source*, that specifies to what database the report applies; and one or more *data field names*, essentially an enumeration of all the names (as represented in the database) of data fields the report returns information on. The *subject* and *attributes* together form a high-level representation of a report's main substance. The *data field names* provide finer detail to the substance. The *source* specification establishes the context of the report (for instance, if it applies to the database of 'parts', or to the 'contracting' database). All available reports are represented in the internal knowledge base using these components.

With the above representation of reports in mind, we can now turn to determining operators for modifying the partial description. We see that we need operations to modify each component individually. Since we have not chosen to impose a structure on the individual members of a component (i.e., it is not possible to say that one particular subject is related to another subject in some way), structural operators such as *generalize* and *specialize* have no meaning. Instead, we define only the basic editing operators of adding, deleting, and replacing for each component. We also define a copy operator for copying portions of example report results into the description.

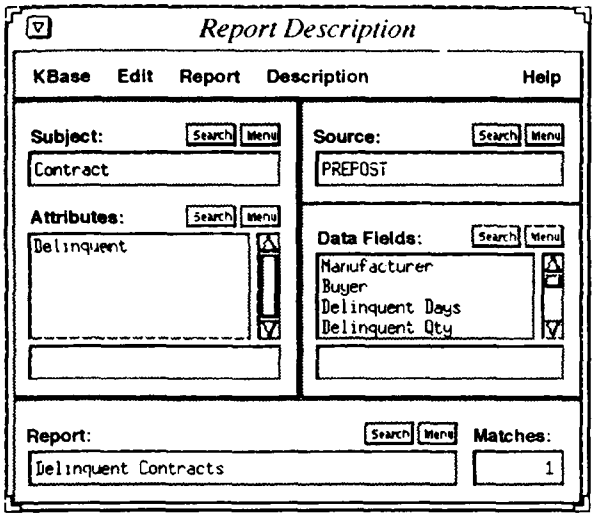

Figure 2: Report Description Window

IMPLEMENTATION

As we mentioned earlier, well designed graphical user interfaces by themselves serve mainly to narrow the syntactic portion of the gap in metaknowledge between expert, and inexperienced users of information systems. The semantic gap is bridged by the underlying clarification dialog. We are now ready to look at how our design for a clarification dialog, as described in the preceding section, has been implemented in BB386 so that both the semantic and syntactic portions of the metaknowledge gap are reduced as much as possible.

Figure 2 shows the main window of the BB386 program, the report description window. It holds a menu bar of commands, an interaction area where the partial description of a report is constructed, and a display area showing the name of one of the reports that matches the current description (it is chosen from the list of matching reports, see Figure 4, below). The report description shown in Figure 2 is fairly detailed (it is seldom the case a description need have this level of detail in order to find a particular report). The values for the *subject* and *attributes* components together specify we want to find reports about "contracts which are delinquent" (there is an implicit AND between all description components). The *source* component specifies the name of the database system we wish to find a report for. Values for each of the data fields specified in the *data field names* component must be present in the report output.

Description component values can be entered in any order. This makes it possible for users to begin by specifying any value(s) they already know, and proceed to refine the description from there. Any changes users make to the description are immediately reflected in the list of matching reports (see Figure 4, below). This tight feedback loop between description modification and matching reports is made possible by h. ' ' an internal knowledge base to work with, instea. f having to perform database queries after each refinement operation.

Users modify the description in one of three ways: by directly typing in new values into their respective type-in fields, by copying values from examples (see below), or by selecting values from menus available for each component. The buttons labeled MENU in Figure 2 cause these menus to be displayed. Figure 3 shows the menu for the *subject* component.

These menus are part of the inner feedback loop shown in Figure 1. They are constrained to show only values that are valid completions to the partial description as it currently appears. For instance, in the description shown in Figure 2, if we had not yet selected a *subject*, but had provided the *attribute* value `delinquent`, the subject menu would contain only the entry `contract` for us to use since it is the only *subject* to which `delinquent` can apply (due to the structure of AFLC's reports). The menus also act to show users what values each component can ' ive. Users are similarly constrained to typing in only valid values for the description components. BB386 provides a substring search mechanism that allows users to type in only partial values for components, and use the buttons labeled SEARCH, in Figure 2, to find completions for the these values.

The bottom-most section of the description window shown in Figure 2 displays the name of one of the reports matching the description, selected from the list of matching reports. This field also acts as a type-in area for

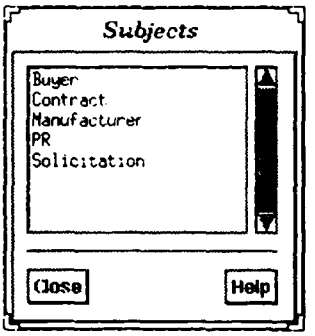

Figure 3: Component menu for *subjects*.

report names. Using this field, users may perform string searches for reports based on their names in the same fashion as the string search facility available for description components. The search applies to the list of matching reports, as it currently appears.

Figure 4 shows the window containing the list of matching reports. It consists of two separate scrolling areas. The upper area is the list of matching reports. Changes to the partial description are immediately

reflected in the contents of this list. The lower area lists examples available for reports selected in the upper list. These examples show 'canned' example of output generated by the report. The names used for examples can reflect particular formatting options, or the values of other parameters the corresponding reports require.

The example window is shown in Figure 5. It presents output from reports as they would appear on paper, or on a terminal screen. Any number of such outputs can be displayed simultaneously for different reports (each will appear in a separate example window). Using the COPY button, users may copy items from the example, such as data field names, into the description window (there is a corresponding PASTE command in the EDIT menu of the description window menu bar). In our current implementation of BB386, this copy facility is based on copying text from the example into the description. There is currently no notion of objects, or object structure, within examples – they are merely text. In future versions of BB386 we plan to implement a more sophisticated copying scheme.

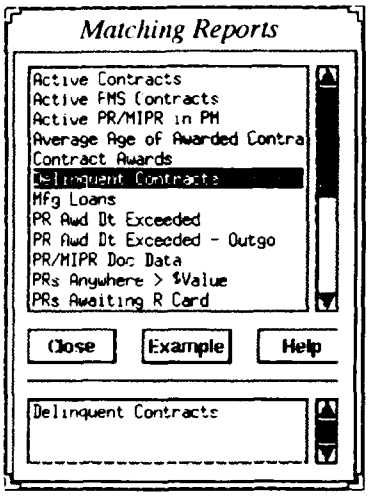

Figure 4: Matching Reports Window

Accelerators

All menus and lists in BB386 are equipped with accelerators to allow experienced users to more rapidly navigate through them. For instance, double-clicking on a report in the list of matching reports (Figure 4) will copy the definition of that report (expressed in terms of *subject, attributes, source,* and *data fields)* into the description area. This facility makes it easy to browse through the reports to examine their components, or to start the reformulation process with a known report and then generalize the description from there.

Selecting a Report

Once the desired report has been found, retrieving it is simply a matter of choosing it from the list of matching reports (Figure 4) and selecting a command from the REPORT pull-down menu shown in Figure 2. After prompting the user for parameters required to generate the report, a request is sent off to the appropriate database system. In the current implementation of BB386, the report results are displayed in a terminal emulation window, detached from the BB386 program. In future implementations we will better integrate this portion of the report retrieval process into BB386.

Knowledge Base Editor

Since BB386 uses an internal knowledge base to do its reasoning with, we need some means of keeping this knowledge base up to date. New reports might need to be added; old, no longer useful reports can be discarded, etc. We also need some way of easily creating entirely new knowledge bases for new applications. To fulfill these requirements we created a "system administrator's mode" in BB386. While in this mode, a variant of the same user interface presented in Figure 2 is used to edit the knowledge base. New reports can be defined, report component values (*subjects, attributes,* etc.) can be added, renamed, or deleted. Newly created reports are immediately merged into the knowledge base, and are available for use when BB386 is returned to its normal

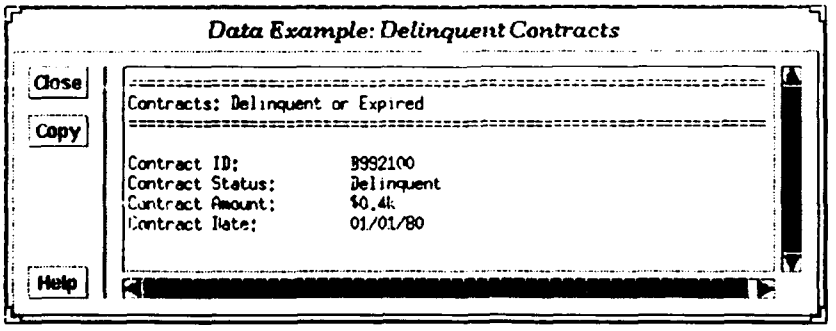

Figure 5: Example Window

mode of operation. To protect the system from accidents caused by inexperienced users, it is possible to password-protect the maintenance mode.

CONCLUSION

BB386 illustrates a further step in our efforts to establish a general procedure for mapping the essential components of the specification by reformulation paradigm onto the user interface dialog requirements of an application. Our goal is to capture this procedure and use it as the basis for a specification by reformulation shell, that will serve as a tool to allow us to more easily instantiate the paradigm for new application domains.

Implementing BB386 has emphasized two important aspects of customizing the specification by reformulation paradigm for new applications, both of which are particularly relevant for our work on constructing a shell. The first is the importance of being able to adapt the operators used to refine the partial description, and to generate feedback based on the description. For BB386 we found that these operations required an internal knowledge base representation of the AFLC's reports. There are many other applications where such an approach might also be necessary: on-line help systems, command selection guidance, geographical map navigation (a potentially interesting, graphical application that could be used with the paradigm) are just a few examples. Clearly, a shell would also need to support such a concept. Related to this matter is the need for adapting the forms of advice given to users, exemplified by the component menus of BB386, which are constrained to show only items that are valid additions to the description. A shell would have to provide hooks for application-specific advice generators to support this notion.

The second important aspect to consider when customizing the paradigm is adapting the various screen presentations of its key components to the application. Model-based user interface design environments, such as the Humanoid system [8], can greatly reduce the amount of effort involved in performing such adaptations. Other methods for providing syntactic guidance are also important. In BB386 these are exemplified by the component item string search facilities.

High-level dialog based interface paradigms, such as specification by reformulation, are important research topics. As user interfaces become more sophisticated in terms of appearances and functionality, the relevance of dialog based paradigms in bridging the final, remaining, semantic gap in users' metaknowledge of systems will become increasingly apparent. Explicitly representing dialog games in applications enables users to establish a degree of confidence in the systems. They know what operations they can expect the applications to support, and from what vantage point their inputs will be interpreted. This should allow users to more rapidly become acquainted with new systems, and to find the most efficient methods of accomplishing their work.

ACKNOWLEDGMENTS

The authors would like to thank Kenneth Mellott from the Air Force Logistics Command, without whose valuable help and guidance BB386 would never have been possible. Thanks also go to Pedro Szekely, Brian Harp, David Benjamin, and Ping Luo for their comments on previous versions of this paper. This research was supported by DARPA through Contract Numbers NCC 2-719 and N00174-91-0015.

REFERENCES

1. Erickson, T., Salomon, G. Designing a Desktop Information System: Observations and Issues. In *Proceedings of CHI '91, Human Factors in Computing Systems*, ACM, 1991, pp. 49-54.
2. Fischer, G., Henninger, S., Redmiles, D. Intertwining Query Construction and Relevance Evaluation. In *Proceedings of CHI '91, Human Factors in Computing Systems*, ACM, 1991, pp. 55-62.
3. Fischer, G., Nieper-Lemke, H. HELGON: Extending the Retrieval by Reformulation Paradigm. In *Proceedings of CHI '89*, ACM, 1989, pp. 357-362.
4. Fischer, G., Stevens, C. Information Access in Complex, Poorly Structured Information Spaces. In *Proceedings of CHI '91*, ACM, 1991, pp. 63-70.
5. Levin, J. A., Moore, J. A. Dialogue Games: Meta-communication Structures for Natural Language Interaction. Technical Report ISI RR-77-53, USC/Information Sciences Institute, 1977.
6. Mann, W. Dialogue Games. Technical Report ISI RR-79-77, USC Information Sciences Institute, 1979.
7. Stefik, M., Williams, M. D. Specification by Reformulation: An Approach to Knowledge Based Interface Design. Intellicorp, Mountain View, CA, 1986.
8. Szekely, P., Luo, P., Neches, R. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings of CHI '92*, ACM, 1992, pp. 507-515.
9. Tou, F. N., Williams, M. D., Fikes, R., Henderson, A., Malone, T. RABBIT: An Intelligent Database Assistant. In *Proceedings of AAAI-82*, 1982, pp. 314-318.
10. Williams, M. D., Hollan, J. D. The Process of Retrieval From Very Long Term Memory. *Cognitive Science #5*, 1981, pp. 87-119.
11. Yen, J., Neches, R., DeBellis, M., Szekely, P., Aberg, P. BACKBORD: An Implementation of Specification by Reformulation. In *Intelligent User Interfaces*, Sullivan, J. W., Tyler, S. W. (Eds.), ACM Press, 1991, pp. 421-444.